Compensation in Collaborative Editing

Stéphane Weiss, Pascal Urso and Pascal Molli
LORIA
Campus Scientifique, BP 239
F-54506 Vandoeuvre-lès-Nancy, France
{weiss,urso,molli,oster}@loria.fr

ABSTRACT

Undo/Redo has been recognized as an important feature of collaborative editing systems. The undo mechanism allows any user to undo any edit operation at any time. Preserving consistency of shared data with the undo feature is a complex issue. Several algorithms exist for managing Undo in collaborative editing, but they are still open issues concerning correctness or intrinsic limitations of these algorithms. Therefore, in this paper we present an novel undo approach in the context of Operational Transformation (OT) framework. We prove the correctness of our solution, we leverage some limitations of current Undo approaches and we indicate how it can be combined with existing OT integration algorithms to provide a complete usable system.

INTRODUCTION

Undo/Redo has been recognized as an important feature of collaborative editing systems [1, 2, 3]. In collaborative systems, the most general model of undo mechanism allows any user to undo any edit operation at any time. Preserving consistency of shared data with the undo feature is a complex issue. Several Undo algorithms have been published within the Operational Transformation (OT) [3, 4] framework.

Ressel proposed the first undo algorithm with Adopted [5]. However, this undo algorithm does not allow the user to undo any operation. The user can only undo his proper operations in the inverse order of their generation.

Next, the ANYUNDO approach [6] provides a mechanism to undo any operations. However it requires the verification of the properties TP1, TP2 and IP1. Unfortunately, transformation functions satisfying these properties have never been published. The undo algorithm from [7] suffers from the same limitations as the ANYUNDO approach.

The COT approach [8] has been designed to simplify the ANYUNDO approach. The major changes presented in COT include, along the integration algorithm, the usage of context vectors and the breaking of property TP2. In this paper, we demonstrate that property TP2 is not broken in the COT approach. We also point out that context vectors grow linearly with the number of undo performed during the editing session. Thus, context vectors limit the number of operation that we can undo. Consequently, the number of operation a user can undo depend on the number of participants and the network's bandwidth.

In this paper, we present the *compensation* approach. This approach does not require context vectors and property IP1. Moreover, compensation can be used with all integration algorithms such as adOPTed, SOCT2, GOTO and COT. It makes the undo feature available even for algorithms with no native undo support such as SOCT2 and SOCT4.

As a proof of concept, we applied the compensation framework to the Tombstone Transformation Functions (TTF) [9]. The correction of the obtained model is proved formally by the automated proof environment VOTE [10].

Based on this approach, we build the Graveyard real-time editor prototype. Graveyard combines a SOCT2 algorithm that does not provide a native undo feature, with TTF transformation functions extended with compensation operations.

COMPENSATION IN THE OT APPROACH

Existing undo methods for OT enforce the document to return to a previous state after the execution of an undo operation. The compensation approach does not require the system returns to a previous state but only to a state semantically equal to a previous state. However, this semantically equal state can also be syntactically equal to the previous state.

We call $C(op)$ the operation which compensates op. The execution of the operation $C(op)$ undoes, from a semantic point of view, the effect of the operation op. $C(op)$ is not necessary the inverse operation of op but it is defined in order to compensate op just after the execution of op. We need to define each operation $C(op)$ in such a way that $S \circ op \circ C(op)$ is a state where the effect of op has been semantically undone.

However, the operation $C(op)$ does not take account of the effect of operations executed between the execution of op and it's compensation by the user. We need to compute an operation $C(op)'$ which realize the effect of $C(op)$ on the current state. To compute the operation $C(op)'$, we need the following algorithm (also illustrated by the Figure 1).

1. First, we determine the operation $C(op)$ which compensates op. $C(op)$ should have compensated op if op was the last operation executed on the current site.

2. We use the forward transformation functions in order to transform $C(op)$ with all operations which have already been executed on this site. The resulting operation is called $C(op)'$. $C(op)'$ is defined on the current state and will be send to other sites where it will be integrated as any other new operation.

This algorithm is known as the naive algorithm for undo [7].

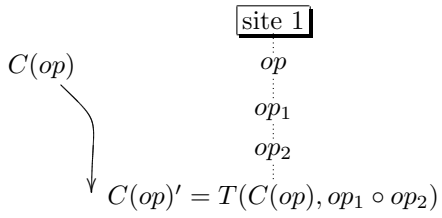

Figure 1. Algorithm of the compensation

CORRECTNESS OF THE COMPENSATION APPROACH

The compensation approach does not depend of the integration algorithm. It is defined at the transformation functions level. In order to be correct, the transformation functions defined for regular and compensation operations have to ensure the following properties.

- The properties $TP1$ and $TP2$ are defined to ensure consistency. Depending on the OT integration algorithm, both properties are required or only $TP1$. Indeed, in our approach, integration algorithms make no distinction between compensating operations and other operations. Then, compensating operations have also to satisfy the correctness properties.

- The property TP_C ensures the respect of the compensation. This property ensures that the compensation effect will always be the same, even if the operation compensated is not the last executed one. This condition is similar to the conditions $C4$ [7] and $IP3$ [8, 11].

$$TP_C : T(C(op), T(seq, op)) = C(T(op, seq))$$

Figure 2 explains the TP_C property. Two sites make concurrent operations. Site1 generates op while site 2 generates a sequence of operations seq. Both sites receive remote operations, transform and integrate them. Now, they are on the same state. Consequently, if they want to compensate the same operation on the same state, they must obviously generate the same operation. Site1 generate $C(op)$ and transform it through following operations $T(seq, op)$. Site2 compensate the last received operation which is $T(op, seq)$. These two compensating operations are defined on the same state, they compensate the same operation, so they must be the same. The verification of this property ensures that whenever an operation is compensated, the compensation effect remains the same.

So, there are three – or two – properties to verify in order to ensure a correct OT system with compensation. Due to their conciseness, these properties are theoretically easy to prove. However, one of the particularity of the OT approach is the huge numbers of cases to check. In such conditions, a hand proof is error-prone, and many transformation functions supposed hand-proven finally revealed themselves false (all counter examples can be found in [9]). On another hand, each of the cases to check can be easily handle by an automated formal theorem prover. Consequently, we choose to use the proof environment VOTE [10] based on the theorem prover Spike [12] which generates all cases and ensures the verification of all required properties.

COMPENSATION IN THE TTF APPROACH

The TTF approach is divided in two parts: the model and the transformation functions. A detailed explanation of the TTF approach and its correctness can be found in [9].

In the TTF approach, deleted characters are kept as tombstones. The document's view only shows visible characters and tombstones are hidden. Consequently, the model differs from the view. Assume that a document is in a state "abcd". Now, a user deletes the character 'b'. In the TTF model, the character is replaced by a tombstone (i.e., the character with a visibility flag set to false). The view differs from the model as the view only contains "acd" while the model contains "a$\not$bcd". Since tombstones are necessary to achieve consistency, they cannot be removed and thus, the operation "Ins" is not inversible.

As the TTF approach is defined for two operations: "Ins(p, c, sid)" and "Del(p, sid)", we need to find two operations to compensate them. The compensating effect required is to obtain that visible characters are the same.

Fortunately, it is obvious that there is no difference between compensating an insertion and deleting a character. So we can use the operation "Del" for compensating "Ins" since it has the desired effect which is removing the character in the user's view.

However, compensating the operation "Del" with the operation "Ins" inserts a new character in spite of the existing tombstone for this character. Thus, we define a new operation "Undel(position, sid)" to compensate the operation "Del". This operation replaces the tombstone of a deleted character by the original character. This operation "Undel" allows to build transformation functions which satisfy TP_C and to reuse tombstones.

The function $C(op)$ links normal operations to compen-

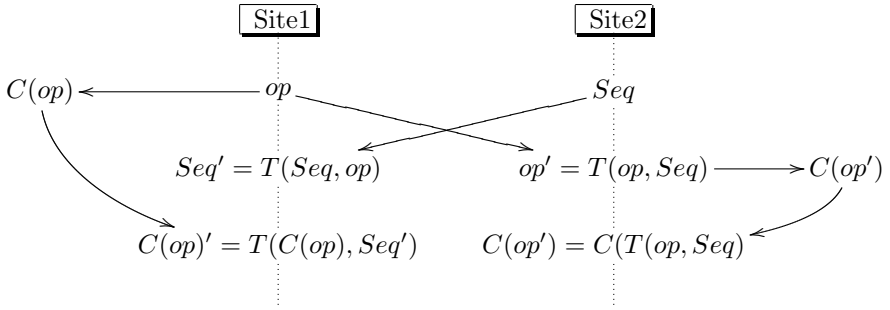

Figure 2. Respect of the compensation effect

sating operations. As we have defined compensating operations, we can now write the function $C(op)$.

```
C(op):
IF op = Ins(p, c, sid) THEN C(op) := Del(p, sid)
IF op = Del(p, sid) THEN C(op) := Undel(p, sid)
IF op = Undel(p, sid) THEN C(op) := Del(p, sid)
```

Finally, we can write the transformation functions for all operations. The definition of the transformation functions for the operations "Ins" and "Del" are the same as presented in [9].

```
T( Ins(p1, c1, sid1),Undel(p2, sid2)):
   return Ins(p1, c1, sid1)
end

T( Del(p1, sid1), Undel(p2, sid2)):
   return Del(p1, sid1)
end

T( Undel(p1, sid1), Ins(p2, c2, sid2)):
   if (p1 < p2) then return Undel(p1, sid1)
   else return Undel(p1 + 1, sid1)
 end

T( Undel(p1, sid1), Undel(p2, sid2)):
   return Undel(p1, sid1)
end

T( Undel(p1, sid1), Del(p2, sid2)):
   return Undel(p1, sid1)
end
```

Correction of the approach

In the TTF approach, the transformation functions are written in order to satisfy the property $TP2$. The property $TP1$ is defined by a state equality, then we have to define the effect of the operation "Undel" on the state. If the effect of "Undel" is simply to make the character visible, the property $TP1$ is violated (see Figure 3)

To ensure $TP1$, we define the effect of "Undel" replacing the visibility flag associated to characters by a visibility level. This visibility level is an integer. Initially, a character inserted has a visibility level of 1. Each time an operation deletes this character, its visibility level is decreased. Each time an operation undeletes this character, we increase its visibility level. A character is said

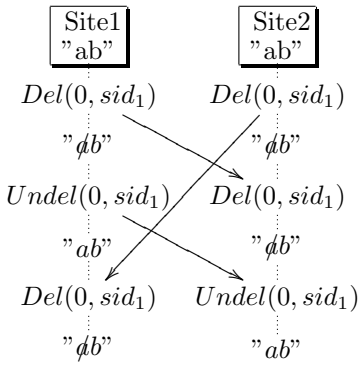

Figure 3. Violation of the property $TP1$

"visible" and appears in the document's view if its visibility level is at least 1. Similarly, a character is said "invisible" and does not appear in the document's view if its visibility level is less than 1 (see figure 4).

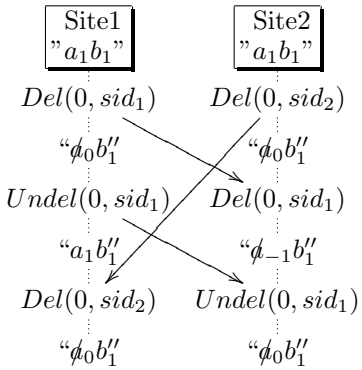

Figure 4. Visibility level

Using the proof environment VOTE [10], we have proven that our transformation functions satisfy the properties $TP1$, $TP2$ and TP_C. The system specification given to the theorem prover Spike can be reviewed and tested at the following url : `http://potiron.loria.fr/projects/graveyard`.

Implementation

In order to validate our approach, we have built the Graveyard prototype. Graveyard is a real-time collaborative text editor. It relies on the SOCT2 algorithm for integrating concurrent operations. SOCT2 does not provide natively undo capabilities. We used the TTF transformation functions with related compensation operations to obtain a real-time collaborative with an undo feature. The general architecture of graveyard is described in figure 5.

For this implementation, we used SOCT2 but we can replace SOCT2 by SOCT4, adOPTed or COT and obtain the same result. Indeed, our framework only introduces the compensation algorithm. To compensate an operation op, we transform $C(op)$ with all operations which have been executed after op. Fortunately, the main goal of every integration algorithms is to transform an operation against a set of concurrent operations. Thus, every OT system already contains such a feature : *Translate Request* for adOPTed, *Transpose Forward* for SOCT4 or *COT-DO* for COT. Finally, the compensation approach can be used any integration algorithm.

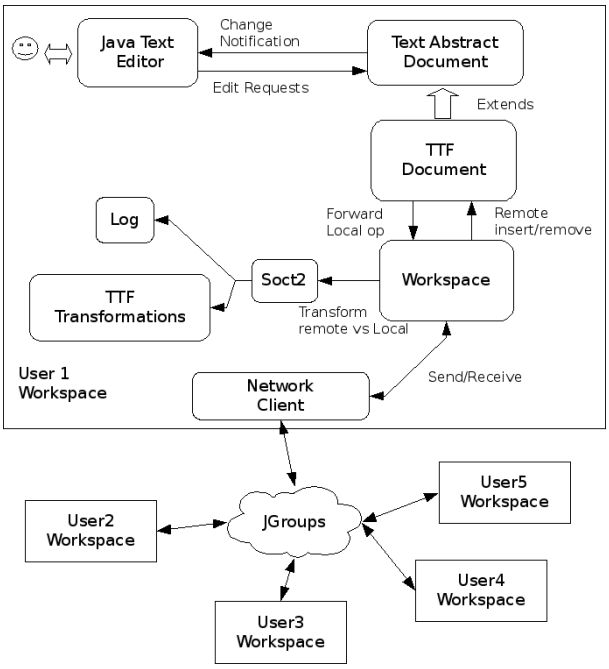

Figure 5. Graveyard architecture

RELATED WORK

In [5], the authors present an undo specific to the adOPTed algorithm. Unfortunately, this solution cannot allow to undo any operation at anytime.

The ANYUNDO algorithm [6] is associated with the GOTO integration algorithm. The GOTO-ANYUNDO approach needs transformation functions which satisfy three properties $TP1$, $TP2$ and $IP1$. The property $IP1$ illustrates the neutrality of do-undo pairs toward the document state. Unfortunately, transformation functions satisfying all these properties have never been published. For instance, TTF functions satisfy $TP1$ and $TP2$ but not $IP1$.

In [7], the authors define two properties $C3$ and $C4$ which are similar to $IP2$ and $IP3$. To ensure the verification of these two properties, the authors introduce a specific operation "undo(op)". This approach defines generic transformation functions for this operation "undo(op)" using the proposed transformation functions. Unfortunately, as the ANYUNDO, a property similar to $IP1$ is required. Therefore, this approach cannot be instantiate as it required properties which have never been published.

In COT [8], the authors propose an approach which requires only $TP1$. To ensure consistency, concurrent operations are ordered before transforming remote operations. Unfortunately, this ordering is not sufficient as shown by Figure 6. Let's assume that concurrent operations order is op_1, op_2 and op_3. When $site3$ receives op_1, due to concurrent operations order, op_1 is transformed with $op_2 \circ op'_3$ with $op'_3 = T(op_3, op_2)$. Similarly, $site4$ transforms op_2 with the sequence $op_1 \circ op''_3$ with $op''_3 = T(op_3, op_1)$. After the execution of op'_1 and op''_2, $site3$ and $site4$ diverge. The property $TP1$ and the same transformation order are not sufficient to ensure consistency.

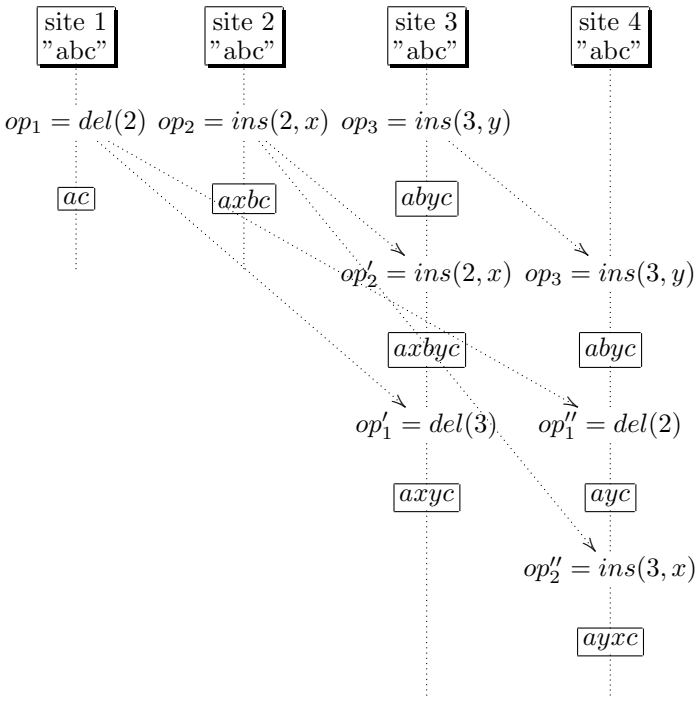

Figure 6. COT Counter-example.

As the property TP2 is not broken, we need transformation function satisfying TP1 and TP2. Therefore, we apply the TTF functions with the COT algorithm (Figure 7)[1]. The TTF function does not satisfy the

[1]The site identifier sid is omited in this example since it

properties IP2 and IP3, but the COT approach aims to enforces them. In figure 7, when $site1$ receives op_2, it has to transform this operation with $op_1 \circ op_3$. As this sequence is a do-undo-pair, the COT algorithm skips this sequence and it does not transform op_2. Using TTF with COT does not ensure consistency in case of undo. Thus, the COT approach fails to enforce IP2 in some cases. And finally, none of the published transformation functions can be directly used with COT to build a text editor with undo capability.

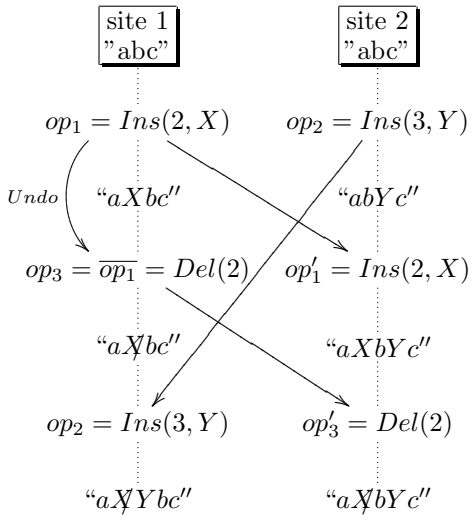

Figure 7. Using TTF functions with COT.

Moreover, the COT approach introduces the notion of context vector. A context vector is composed by a classical state vector and an extra component designed for undo. This extra component stores all undone operations. Each of them is associated with an integer representing how many times this operation has been undone. This context vector is associated to each operation. Its size grows linearly with the number of undone operation. This means that the upload bandwidth will be eventually saturated by the growing context vectors.

To illustrate this issue, let's assume that 10 users are editing the same text document using the COT algorithm. They are connected through Internet using an ADSL connexion. The upload limit for a such connexion is usually $1Mb/s$. We assume that a user generates 4 operations per second. If $32bits$ integer are used, the bandwidth is saturated by context vectors after 4091 undo[2]. Consequently, each user can only undo around 400 operations and, thus, affects the same number of characters.

On another hand, the TTF approach, which is independent of the compensation approach, the document model keeps the tombstones and is thus also unbounded. However, the operations sent on the network have a fixed size and the bandwidth consumption is limited.

Finally, due to the space complexity of context vector, we consider that the COT approach only supports few users connected on an high bandwidth network.

CONCLUSIONS

Many approaches aimed to provide an undo mechanism which allows a user to undo any operations at any time. Unfortunately, most of them are not instantiable and the other one generates heavy traffics which are not compatible with real-time constraints.

In this paper, we introduced our compensation mechanism. The compensation approach is more generic than existing undo approach: we can apply compensation to all transformation functions even if some operations have no inverse. An important feature of our approach is that the resulting transformation functions remain generic towards integration algorithms. Consequently, we can apply these functions with COT, SOCT2, SOCT4, GOTO and adOPTed. We have a complete solution to build text editors with undo capabilities. The compensation approach proposed in this paper has been implemented in the Graveyard collaborative text editor based on the tombstone transformation approach.

In future works, we will implement more existing integration algorithms in the Graveyard prototype. Thus, we will be able to benchmark the different integration algorithms.

REFERENCES

1. G. D. Abowd and A. J. Dix, "Giving undo attention." *Interacting with Computers*, vol. 4, no. 3, pp. 317–342, 1992.
2. R. Choudhary and P. Dewan, "A general multi-user undo/redo model." in *ECSCW*, 1995, pp. 229–246.
3. C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems," *TOCHI*, vol. 5, no. 1, pp. 63–108, Mars 1998.
4. C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems." in *SIGMOD Conference*, J. Clifford, B. G. Lindsay, and D. Maier, Eds. ACM Press, 1989, pp. 399–407.
5. M. Ressel and R. Gunzenhäuser, "Reducing the problems of group undo." in *GROUP*, 1999, pp. 131–139.
6. C. Sun and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems," *TOCHI*, vol. 9, no. 1, pp. 1–41, Mars 2002.
7. J. Ferrié, N. Vidot, and M. Cart, "Concurrent undo operations in collaborative environments

does not affect the result.

[2]We only count context vectors without any encapsulations such as IP,TCP,...

using operational transformation." in *CoopIS*, ser. Lecture Notes in Computer Science, vol. 3290. Springer, Novembre 2004, pp. 155–173.

8. D. Sun and C. Sun, "Operation Context and Context-based Operational Transformation," in *CSCW*. Banff, Alberta, Canada: ACM Press, Novembre 2006, pp. 279–288.

9. G. Oster, P. Urso, P. Molli, and A. Imine, "Tombstone transformation functions for ensuring consistency in collaborative editing systems," in *CollaborateCom*. Atlanta, Georgia, USA: IEEE Press, November 2006.

10. A. Imine, P. Molli, G. Oster, and P. Urso, "Vote: Group editors analyzing tool: System description." *Electr. Notes Theor. Comput. Sci.*, vol. 86, no. 1, 2003.

11. C. Sun, "Undo any operation at any time in group editors." in *CSCW*, 2000, pp. 191–200.

12. S. Stratulat, "A general framework to build contextual cover set induction provers." *J. Symb. Comput.*, vol. 32, no. 4, pp. 403–445, September 2001.